

Fast and Simple Relational Processing of Uncertain Data

Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu

Saarland University Database Group

Saarbrücken, Germany

{lublena, jansen, koch, olteanu}@infosys.uni-sb.de

Abstract

This paper introduces U-relations, a succinct and purely relational representation system for uncertain databases. U-relations support attribute-level uncertainty using vertical partitioning. If we consider positive relational algebra extended by an operation for computing possible answers, a query on the logical level can be translated into, and evaluated as, a single relational algebra query on the U-representation. The translation scheme essentially preserves the size of the query in terms of number of operations and, in particular, number of joins. Standard techniques employed in off-the-shelf relational database management systems are effective for optimizing and processing queries on U-relations. In our experiments we show that query evaluation on U-relations scales to large amounts of data with high degrees of uncertainty.

1 Introduction

Several recent works [10, 9, 8, 2, 14, 4, 6] aim at developing scalable representation systems and query processing techniques for large collections of uncertain data as they arise in data cleaning, Web data management, and scientific databases. Most of them are based on a possible worlds semantics, and for all of them such a semantics can be conveniently defined.

Four desiderata for representation systems for incomplete information appear important.

1. Expressiveness. The representation should be closed under the application of (relational algebra) queries and data cleaning algorithms (which remove some possible worlds). That is, the results of such operations to the represented data should be again representable within the formalism.

2. Succinctness. It should be possible to represent large sets of alternative worlds using fairly little space.

3. Efficient query evaluation. A trade-off is required between the succinctness of a representation formalism and the complexity of evaluating interesting queries. This trade-off follows from established theoretical results [1, 11, 6].

However, while the formalisms in the literature tend to differ in succinctness, several have polynomial-time data complexity for (decision) problems such as tuple possibility under *positive* (but not full) relational algebra. This includes v-tables [12, 11], uncertainty-lineage databases (ULDBs) [8], and world-set decompositions (WSDs) [6].

4. Ease of use for developers and researchers in the sense that the representation system can be easily put on top of a relational DBMS. This in particular includes that queries on the logical schema level can be translated down to, ideally, relational algebra queries on the representation relations and that this translation is simple and easy to implement. This goal is motivated by the availability and maturity of existing relational database technology.

An important aspect of a representation system is whether it represents uncertainty at the *attribute-level* or the *tuple-level*. Attribute-level representation refers to the succinct representation of relations in which two or more fields of the same tuple can independently take alternative values (see also [6]). Attribute-level representation of uncertainty (as supported by c-tables [12] and WSDs) offers finer granularity of independence than tuple-level approaches such as [8, 10, 2]. This is useful in applications like data cleaning in which the values of several fields of a single tuple can be independently uncertain. For instance, the U.S. Census Bureau maintains relations with dozens of columns (> 50), most of which may require cleaning [4].

U-relations. In this paper, we develop and study *U-relations*, a representation system that we introduce with the following example.

Example 1.1. Let us assume that an aerial photograph of a battlefield shows four vehicles at distinct positions 1 to 4. The resolution of the image does not allow for the identification of vehicle types, but we can draw certain conclusions from earlier reconnaissance and a calculation of the maximum distance each vehicle may have covered since. Say we know that vehicle 1 is (a) a friendly tank. Vehicles 2 and 3 are (b) a friendly transport and (c) an enemy tank, but we do not know which one is which. Nothing is known about

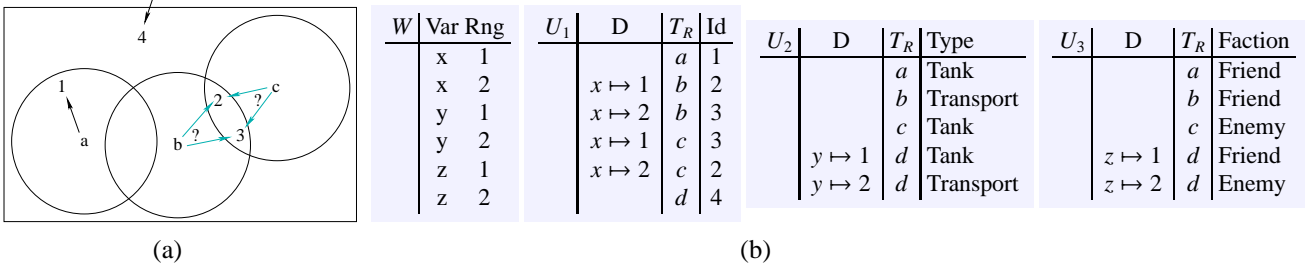


Figure 1. Map with moving vehicles (a) and U-relational database representation of the possible worlds at the time the aerial photograph detecting vehicles 1,2,3,4 was taken (b).

vehicle 4. Figure 1a shows a schematic drawing of how this scenario can arise. Only 1 is in the range of (a); 2 and 3 are in the ranges of (b) and (c); and position 4 is near the border of the photograph but outside the ranges of (a), (b), and (c), so this vehicle must have newly moved onto the map.

We want to model this by an uncertain database of schema $R(\text{Id}, \text{Coord}, \text{Type}, \text{Faction})$, representing the ids (1–4), coordinate positions, types, and factions of the vehicles on the map. Let us assume there are only two vehicle types (tank or transport) and two factions (friend or enemy). Then there are eight possible worlds. We obtain one by taking three choices – answering the following questions: Has the friendly transport (b) now become vehicle 2 ($x \mapsto 1$) or 3 ($x \mapsto 2$)? Is vehicle 4 a tank ($y \mapsto 1$) or a transport ($y \mapsto 2$)? Is vehicle 4 friendly ($z \mapsto 1$) or an enemy ($z \mapsto 2$)? Thus the uncertainty can be naturally modelled using three variables x, y, z that each can independently take one of two values.

We model this scenario by the U-relational database shown in Figure 1b. We use vertical partitioning (cf. e.g. [7, 15]) to achieve attribute-level representation. R is represented using four U-relations, one for each column of R . The U-relation for the coordinate positions (which are all certain) is not shown since we do not want to use it subsequently, but of course, conceptually, coordinate positions are an important feature of the example and have to be part of the schema. In addition there is a relation W which defines the possible values the three variables can take.

We can compute a vertical decomposition of one world given by a valuation θ of the variables x, y, z by (*) removing all the tuples from the U-relations whose D columns contain assignments that are inconsistent with θ (For example, if $\theta = \{x \mapsto 1, y \mapsto 1, z \mapsto 1\}$ then we remove the third and fifth tuples of U_1 and the fifth tuples of U_2 and U_3 .) and then (*) projecting the D columns away. Of course we can resolve the vertical partitioning by joining the decomposed relations on the tuple id columns T_R . \square

U-relations have the following properties:

- **Expressiveness:** U-relations are *complete* for finite sets of possible worlds, that is, they allow for the representation of any finite world-set.
- **Succinctness:** U-relations represent uncertainty on the attribute level. Even though they allow for more efficient

query evaluation, U-relations are, as we show, exponentially more succinct than ULDBs and WSDs. That is, there are (relevant) world-sets that necessarily take exponentially more space to represent by ULDBs or WSDs than by U-relations.

- **Leveraging RDBMS technology:** U-relations allow for a large class of queries (positive relational algebra extended by the operation “possible”) to be processed *using relational algebra only*, and thus efficiently in the size of the data. Our approach is the first so far to achieve this for the above-named query language. Indeed, this not only settles that there is a succinct and complete *attribute-level* representation for which the so-called tuple Q-possibility problem for positive relational algebra is in polynomial time (previously open [6]) but puts a rich body of research results and technology at our disposal for building uncertain database systems.

This makes U-relations the most efficient and scalable approach to managing uncertain databases to date.

- **Parsimonious translation:** The translation from relational algebra expressions on the logical schema level to query plans on the physical representations replaces a selection by a selection, a projection by a projection, a join by a join (however, with a more intricate join condition), and a “possible” operation by a projection. We have observed that state-of-the-art RDBMS do well at finding efficient query plans for such physical-level queries.

Ease of use: A main strength of U-relations is their simplicity and low “cost of ownership”:

- The representation system is purely relational and in close analogy with relational representation schemes for vertically decomposed data. Apart from the column store relations that represent the actual data, there is only a single auxiliary relation W (which we need for computing certain answers, but not for possible answers).
- Query evaluation can be fully expressed in relational algebra. The translation is quite simple and can even be done by hand, at least for moderately-sized queries.
- The query plans obtained by our translation scheme are usually handled well by the query optimizers of off-the-shelf relational DBMS, so the implementation of special

operators and optimizer extensions is not strictly needed for acceptable performance.

Thus U-relations are not only suited as a representation system for dedicated uncertain database implementations such as MayBMS [4], but are also relevant to “casual users” of representation systems for uncertain data, such as researchers in data cleaning and data integration who want to store and query uncertain data without great effort.

Apart from those implicitly mentioned above, we make the following further contributions in this paper.

- We study algebraic query optimization and present equivalences that hold on vertically decomposed representations. We address query optimization using them in the context of managing uncertainty with U-relations.
- We present an algorithm for normalizing a U-relational representation obtained from a query. Normalized U-relational databases yield a conceptually simple algorithm for computing the certain answers of queries. In particular, certain answer tuples on normalized tuple-level representations can be computed using relational algebra only, which is not true in general for previous representation systems.
- We provide experimental evidence for the efficiency and relevance of our approach.

The structure of the paper is as follows. Section 2 establishes U-relations formally. Section 3 presents our reduction from queries on the logical level to relational algebra on the level of U-relations and addresses algebraic query evaluation. Section 4 presents the normalization algorithm. Section 5 discusses the relationship between U-relations, WSDs and ULDBs and argues that U-relations combine the advantages of the other two formalisms without sharing their drawbacks. In Section 6, we report on our experiments with U-relations. We conclude with Section 7.

2 U-relational databases

We define *world-sets* in close analogy to the case of c-tables [12]. Consider a finite set of variables over finite domains. A *possible world* is represented by a total valuation (or assignment) $f : \text{Var} \mapsto \text{Rng}$ of variables to constants in their domains, and the world-set is represented by the finite set of all total valuations¹. We represent relationally the variable set and the associated domains by a *world-table* over schema $W(\text{Var}, \text{Rng})$ such that W consists of all pairs (x, v) of variables x and values v in the domain of x .

Example 2.1. The world-table W in Figure 1 defines three variables x, y, z , whose common domain is $\{1, 2\}$. The number of worlds defined by W is $2 \cdot 2 \cdot 2 = 8$. \square

¹This is a generalization of world-set decompositions of [4], where component ids are variables and local world ids are domain values.

Given a world-table W , a *world-set descriptor* over W , or *ws-descriptor* for short, is a valuation \bar{d} such that its graph is a subset of W . If \bar{d} is a *total* valuation, then it represents one world. In our examples, to represent the entire world-set we use an *empty* ws-descriptor, as a shortcut for a singleton ws-descriptor with a new variable with a singleton domain.

We are now ready to define databases of U-relations.

Definition 2.2. A *U-relational database* for a world-set over schema $\Sigma = (R_1[\bar{A}_1], \dots, R_k[\bar{A}_k])$ is a tuple

$$(U_{1,1}, \dots, U_{1,m_1}, \dots, U_{k,1}, \dots, U_{k,m_k}, W),$$

where W is a world-table and each relation $U_{i,j}$ has schema $U_{i,j}[\bar{D}_{i,j}; \bar{T}_{R_i}; \bar{B}_{i,j}]$ such that $\bar{D}_{i,j}$ defines ws-descriptors over W , \bar{T}_{R_i} defines tuple ids, and $\bar{B}_{i,1} \cup \dots \cup \bar{B}_{i,m_i} = \bar{A}_i$.

A ws-descriptor $\{c_1 \mapsto l_1, \dots, c_k \mapsto l_k\}$ is relationally encoded in $\pi_{\bar{D}_{i,j}}(U_{i,j})$ of arity $n \geq k$ as a tuple $(c_1 \mapsto l_1, \dots, c_k \mapsto l_k, c_{k+1} \mapsto l_{k+1}, \dots, c_n \mapsto l_n)$, where each $c_i \mapsto l_i$ is a $c_j \mapsto l_j$ for any j and all i with $1 \leq j \leq k < i \leq n$.

Although we speak of vertical partitioning, we do not require the value columns of $U_{i,j}$ to disjointly partition the columns of R_i . Indeed, overlap may be useful to speed up query evaluation, see e.g. [15].

We next define the semantics of a U-relational database. To obtain a possible world we first choose a total valuation f over W . We then process the U-relations tuple by tuple. If the function f extends² the ws-descriptor \bar{d} of a tuple of the form $(\bar{d}, \bar{t}, \bar{a})$ from a U-relation of schema $(\bar{D}, \bar{T}, \bar{A})$, we insert in that world the values \bar{a} into the \bar{A} -fields of the tuple with identifier \bar{t} . In general this may leave some tuples partial in the end (i.e., the values for some fields have not been provided.) These tuples are removed from the world.

We require, for a U-relational database (U_1, \dots, U_n, W) to be considered valid, that the representation does not provide several contradictory values for a tuple field in the same world. Formally, we require, for all $1 \leq i, j \leq n$, and tuples $t_1 \in U_i[\bar{D}_i, \bar{T}_i, \bar{A}_i]$ and $t_2 \in U_j[\bar{D}_j, \bar{T}_j, \bar{A}_j]$ such that U_i and U_j are vertical partitions of the same relation, that if there is a world that extends both $t_1.\bar{D}_i$ and $t_2.\bar{D}_j$, then for all $A \in (\bar{A}_i \cap \bar{A}_j)$, $t_1.A = t_2.A$ must hold.

Example 2.3. Suppose there are two U-relations with schemata $U_1[\bar{D}_1; \bar{T}_R; A, B]$ and $U_2[\bar{D}_2; \bar{T}_R; B, C]$ that jointly represent columns A, B , and C of a relation R . Assume tuples $(c_1, 1, t_1, a, b) \in U_1$ and $(c_2, 2, t_1, b', c) \in U_2$. Then U_1 and U_2 cannot form part of a valid U-relational database because there would be a world with $c_1 \mapsto 1, c_2 \mapsto 2$ in which the tuple from U_1 requires field $t_1.B$ to take value b while the tuple from U_2 requires the same field to take value b' . \square

A salient property of U-relational databases is that they form a *complete representation system* for finite world-sets.

Theorem 2.4. Any finite set of worlds can be represented as a U-relational database.

²That is, for all x on which \bar{d} is defined, $\bar{d}(x) = f(x)$.

3 Query Processing

The semantics of a query Q on a world-set is to evaluate Q in each world. For complete representation systems like U-relational databases, there is an equivalent, more efficient approach [12]: Translate Q into a query \hat{Q} such that the evaluation of \hat{Q} on a U-relational encoding of the world-set produces the U-relational encoding of the answer to Q .

Queries on vertical decompositions. U-relations rely essentially on vertical decomposition for succinct (attribute-level) representation of uncertainty. To evaluate a query, we first need to reconstruct relations from vertical decompositions by (1) joining two partitions on the common tuple id attributes and (2) discarding the combinations that yield inconsistent ws-descriptors. We call this operation *merge* and give its precise definition in Figure 4, where the two above conditions are defined by α and ψ , respectively.

Example 3.1. Consider the U-relational database of Figure 1. The query $\sigma_{\text{Faction}=\text{'Enemy'} \wedge \text{Type}=\text{'Tank'}}(R)$ lists the enemy tanks on the map. To answer this query, we need to *merge* the necessary partitions of R and obtain a new query with $\text{merge}(\pi_{\text{Faction}}(R), \pi_{\text{Type}}(R))$ in the place of R . \square

Our query evaluation approach can take full advantage of query evaluation and optimization techniques on vertical partitions. First, it does not require to reconstruct the entire relations involved in the query, but rather only the necessary vertical partitions. Second, necessary partitions can be flexibly merged in during query evaluation. Thus early and late tuple materialization [15] carry over naturally to our framework. For this, our *merge* operator allows to merge two partitions not only if they are given in their original form, but also if they have been modified by queries.

The first advantage only holds for so-called *reduced* U-relational databases, which do not have tuples that cannot be completed in any world. That is, each tuple of a reduced U-relation can always be completed to an actual tuple in a world. The advantage becomes evident even for a simple projection query. Consider a reduced database containing a U-relation U defining the A attribute of R . To evaluate $\pi_A(R)$ we do not need to merge in all U-relations defining the attributes of R and later project on A . Instead, the answer is simply U . In the following, we assume that the input database is always reduced. As we will discuss next, our query evaluation technique always produces reduced U-relations for reduced input U-relational databases.

Example 3.2. Consider the following non-reduced database of two U-relations:

| U_1 | D | T | A |
|-------|-----------------|-------|-------|
| | $c_1 \mapsto 1$ | t_1 | a_1 |
| | $c_2 \mapsto 1$ | t_2 | a_2 |

| U_2 | D | T | B |
|-------|-----------------|-------|-------|
| | $c_1 \mapsto 1$ | t_1 | b_1 |
| | $c_1 \mapsto 2$ | t_1 | b_2 |

In each U-relation the second tuple cannot find a partner in the other U-relation with which a complete tuple (with both

$$\text{merge}(\pi_{\bar{X}}(R), \pi_{\bar{A}-\bar{X}}(R)) = R, \quad \text{where } \bar{A} = \text{sch}(R) \quad (1)$$

$$\text{merge}(R, S) = \text{merge}(S, R) \quad (2)$$

$$\text{merge}(\text{merge}(R, S), T) = \text{merge}(R, \text{merge}(S, T)) \quad (3)$$

$$\sigma_{\phi(\bar{X})}(\text{merge}(R, S)) = \text{merge}(\sigma_{\phi(\bar{X})}(R), S) \quad (4)$$

$$\text{where } \bar{X} \subseteq \text{sch}(R)$$

$$\text{merge}(R, S) \bowtie_{\phi(\bar{X}, \bar{Y})} T = \text{merge}(R \bowtie_{\phi(\bar{X}, \bar{Y})} T, S) \quad (5)$$

$$\text{where } \bar{X} \cup \bar{Y} \subseteq \text{sch}(R) \cup \text{sch}(T)$$

$$\pi_{\bar{X}}(\text{merge}(R, S)) = \text{merge}(\pi_{\bar{X} \cap \bar{A}}(R), \pi_{\bar{X} \cap \bar{B}}(S)) \quad (6)$$

$$\text{where } \text{sch}(R) = \bar{A}, \text{sch}(S) = \bar{B}$$

Figure 2. Algebraic equivalences for relational algebra queries with merge operator.

attributes A and B) can be formed. If these second tuples are removed, the database is reduced. \square

We can always reduce a U-relational database as follows: We filter each U-relation using semijoins with each of the other U-relations representing data of the same relation R_i . The semijoin conditions are the α and ψ -conditions.

Proposition 3.3. *Given a schema Σ , there is a relational algebra query that reduces a U-relational database over Σ .*

Algebraic equivalences. Figure 2 gives algebraic equivalences of relational algebra expressions with merge operator on vertical decompositions: Merging is the reverse of vertical partitioning, it is commutative and associative, it commutes with selections, joins, and projections.

Standard heuristics known from classical query optimization for relational algebra apply here as well. Intuitively, we usually push down projections and selections and merge in U-relations as late as possible. An interesting new case is the decision on join ordering among an explicit join from the input query and a join due to merging: If the merge is executed before the explicit join, it may reduce the size of an input relation to join. We have seen in our experiments that the standard selectivity-based cost measures employed by relational database management systems do a good job, as long as the queries remain reasonably small.

Example 3.4. Consider a U-relational database \mathcal{U} that represents a set of possible worlds over two TPC-H relations Ord and Cust (short for Order and Customer, respectively) [16]. \mathcal{U} has one U-relation for each attribute of the two relations, of which we only list DATE and CUSTKEY for Ord, and NAME and CUSTKEY for Cust. The following query finds all dates of orders placed by Al after 2003:

$$\pi_{\text{DATE}}(\sigma_{\text{NAME}=\text{'Al'}}(\text{Cust}) \bowtie_{\text{CUSTKEY}} \sigma_{\text{DATE}>2003}(\text{Ord}))$$

Figure 3 shows three possible plans P1, P2, and P3 using operators on vertical decompositions. The naïve plan

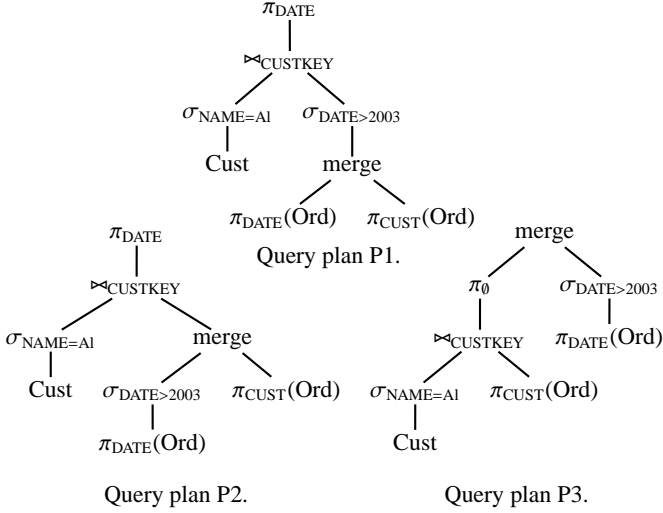


Figure 3. Three equivalent query plans.

P1 first reconstructs Ord from its two partitions then applies the selection and the join with Cust. In P2 and P3 the merge operator is pushed up in the plans, first immediately above the selection (P2), and then above the join operator (P3). Among the three plans, P1 is clearly the least efficient. However, without statistics about the data, one cannot tell which of P2 and P3 should be preferred. If $DATE > 2003$ is very selective, then merging immediately thereafter as in P2 will lead to filtering of tuples from $\pi_{CUSTKEY}(Ord)$ and thus fewer tuples will be processed by the join. Is this not the case, then first merging only increases the number and size of the tuples that have to be processed by the join. Also, in P3 all value attributes except of DATE are projected away after the join as they are not needed for the final result. \square

Queries on U-relations. Figure 4 gives the function $\llbracket \cdot \rrbracket$ that translates positive relational algebra queries with *poss* and *merge* operators into relational algebra queries on U-relational databases.

The *poss* operator applied on a U-relation U closes the possible worlds semantics by computing the set of tuples possible in U . It thus translates to a simple projection on the value attributes of U . The result of a projection is a U-relation whose value attributes are those from the projection list (thus the input ws-descriptors and tuple ids are preserved). Selections apply conditions on the value attributes.

The merge operator that reconstructs a relation from its vertical partitions was already explained. Similarly to the merge, the join uses the ψ -condition to discard tuple combinations with inconsistent ws-descriptors. Figure 4 gives the translation in case U_1 and U_2 do not contain partitions of the same relation. For the case of self-joins we require aliases for the copies of the relation involved in it such that they do not have common tuple id attributes.

The union of U_1 and U_2 like the ones from Figure 4 is sketched next. We assume that $\bar{A}_1 = \bar{A}_2$, $\bar{T}_1 \cap \bar{T}_2 = \emptyset$, and the tuples of different relations have different ids. To

$$\begin{aligned}
 &\text{Let } U_1 := \llbracket Q_1 \rrbracket \text{ with schema } [\bar{D}_1, \bar{T}_1, \bar{A}_1], \\
 &U_2 := \llbracket Q_2 \rrbracket \text{ with schema } [\bar{D}_2, \bar{T}_2, \bar{A}_2], \\
 &\alpha := \bigwedge_{T \in \bar{T}_1 \cap \bar{T}_2} (U_1.T = U_2.T), \\
 &\psi := \bigwedge_{D' \in U_1.\bar{D}_1, D'' \in U_2.\bar{D}_2} (D'.\text{Var} = D''.\text{Var} \Rightarrow D'.\text{Rng} = D''.\text{Rng}). \\
 &\llbracket poss(Q_1) \rrbracket := \pi_{\bar{A}_1}(U_1) \\
 &\llbracket \pi_{\bar{X}}(Q_1) \rrbracket := \pi_{\bar{D}_1, \bar{T}_1, \bar{X}}(U_1), \quad \text{where } \bar{X} \subseteq \bar{A}_1 \\
 &\llbracket \sigma_{\phi}(Q_1) \rrbracket := \sigma_{\phi}(U_1), \quad \text{where } \phi \text{ on } \bar{A}_1 \\
 &\llbracket Q_1 \bowtie_{\phi} Q_2 \rrbracket := \pi_{\bar{D}_1, \bar{D}_2, \bar{T}_1 \cup \bar{T}_2, \bar{A}_1 \bar{A}_2}(U_1 \bowtie_{\phi \wedge \psi} U_2), \\
 &\quad \text{where } \bar{T}_1 \cap \bar{T}_2 = \emptyset \\
 &\llbracket merge(Q_1, Q_2) \rrbracket := \pi_{\bar{D}_1, \bar{D}_2, \bar{T}_1 \cup \bar{T}_2, \bar{A}_1 \bar{A}_2}(U_1 \bowtie_{\alpha \wedge \psi} U_2)
 \end{aligned}$$

Figure 4. Translation of queries with merge into queries on U-relations.

bring U_1 and U_2 to the same schema, we first ensure ws-descriptors of the same size by pumping in the smaller ws-descriptors already contained variable assignments, and add new (empty) columns \bar{T}_2 to U_1 and \bar{T}_1 to U_2 . We then perform the standard union.

From our translation $\llbracket \cdot \rrbracket$ it immediately follows that

Theorem 3.5. *Positive relational algebra queries extended with the possible operator can be evaluated on U-relational databases using relational algebra only.*

Example 3.6. Recall the U-relational database of Figure 1 storing information about moving vehicles. Consider a query asking for ids of enemy tanks:

$$S = \pi_{Id}(\sigma_{Type='Tank' \wedge Faction='Enemy'}(R))$$

After merging the necessary partitions of relation R and translating it into positive relational algebra, we obtain

$$\pi_{Id}(\sigma_{Type='Tank' \wedge Faction='Enemy'}(U_1 \bowtie_{\alpha_1 \wedge \psi_1} U_2 \bowtie_{\alpha_2 \wedge \psi_2} U_3)),$$

where the conditions ψ_1 , ψ_2 , α_1 , and α_2 follow the translation given in Figure 4. The three vertical partitions are joined on the tuple id attributes (α_1 and α_2) and the combinations with conflicting mappings in the ws-descriptors are discarded (ψ_1 and ψ_2). Before and after translation, the query is subject to optimizations as discussed earlier. (In this case, a good query plan would first apply the selections on the partitions, then project away the irrelevant value attributes Type and Faction, and then merge the partitions).

| U_4 | D_1 | D_2 | T_S | Id |
|-------|---------------|---------------|-------|----|
| | $x \mapsto 1$ | | c | 3 |
| | $x \mapsto 2$ | | c | 2 |
| | $y \mapsto 1$ | $z \mapsto 2$ | d | 4 |

The above U-relation U_4 encodes the query answer. \square

Example 3.7. We continue Example 3.6 and ask whether it is possible that the enemy has two tanks on the map, and if so, which vehicles are those. For this, we compute the pairs of enemy tanks as a self-join of S : $(S \ s_1) \bowtie_{s_1.Id \neq s_2.Id} (S \ s_2)$. This query is in turn equivalent to a self-join of U_4 .

| U_5 | D_1 | D_2 | D_3 | T_{s_1} | T_{s_2} | Id_1 | Id_2 |
|-------|---------------|---------------|---------------|-----------|-----------|--------|--------|
| | $x \mapsto 1$ | $y \mapsto 1$ | $z \mapsto 2$ | c | d | 3 | 4 |
| | $x \mapsto 2$ | $y \mapsto 1$ | $z \mapsto 2$ | c | d | 2 | 4 |
| | $y \mapsto 1$ | $z \mapsto 2$ | $x \mapsto 1$ | d | c | 4 | 3 |
| | $y \mapsto 1$ | $z \mapsto 2$ | $x \mapsto 2$ | d | c | 4 | 2 |

The answer is encoded by the above U-relation U_5 . Note that the combinations of the first two tuples of U_4 are not in U_5 , because they have inconsistent ws-descriptors and are filtered out using the ψ -condition (vehicle c cannot be at the same time at two different positions). To obtain the possible pairs of vehicle ids, we apply the poss operator on U_5 . This is expressed as the projection on the value attributes of U_5 . \square

Our translation yields relational algebra queries, whose evaluation always produces tuple-level U-relations, i.e., U-relations without vertical decompositions, by joining and merging vertical partitions of relations. Following the definition of the merge operator, if the input U-relations are reduced, then the result of merging vertical partitions is also reduced. We thus have that

Proposition 3.8. *Given a positive relational algebra query Q and a reduced U-relational database U , $\llbracket Q \rrbracket(U)$ is a reduced U-relational database.*

4 Normalization of U-relations

U-relations do not forbid large ws-descriptors. The ability to extend the size of ws-descriptors is what yields efficient query evaluation on U-relations. However, large ws-descriptors cause an inherent processing overhead. Also, after query evaluation or dependency chasing on a U-relational database, it may happen that tuple fields, which used to be dependent on each other, become independent. In such a case, it is desirable to optimize the world-set representation [6]. We next discuss one approach to normalize U-relational databases by reducing large ws-descriptors to ws-descriptors of size one. Normalization is an expensive operation per se, but it is not unrealistic to assume that uncertain data is initially in normal form [4, 6] and can subsequently be maintained in this form.

Definition 4.1. A U-relational database is normalized if all ws-descriptors of its U-relations have size one.

Algorithm 1 gives a normalization procedure for U-relations that determines classes of variables that co-occur in some ws-descriptors and replaces each such class by one

Algorithm 1: Normalization of ws-descriptors.

Input: Reduced U-relational database $\mathcal{U} = (U_1, \dots, U_m, W)$

Output: Normalized reduced U-relational database.

begin

$R :=$ the relation consisting of all pairs of variables (c_i, c_j) that occur together in some ws-descriptor of \mathcal{U} ;
 $\mathcal{G} :=$ the graph whose node set is the set of variables and whose edge relation is the refl. and trans. closure of R ;
 Compute the connected components of \mathcal{G} ;

foreach U-relation $U_j(D_1, \dots, D_n, \bar{T}, \bar{A})$ of \mathcal{U} **do**

$U'_j :=$ empty U-relation over $U'_j(\text{Var}, \text{Rng}, \bar{T}, \bar{A})$;

foreach $t \in U$ **do**

$G_i :=$ connected component of \mathcal{G} with id i such that the nodes $t.\text{Var}_1, \dots, t.\text{Var}_n$ are in G_i ;

$\{c_{i_1}, \dots, c_{i_k}\} = G_i - \{t.\text{Var}_1, \dots, t.\text{Var}_n\}$;

foreach $l_{i_1} : (c_{i_1}, l_{i_1}) \in W, \dots, l_{i_k} : (c_{i_k}, l_{i_k}) \in W$

do

$/*$ Compute a new domain value ($f_{|G_i|}$ is either the identity or better, for atomic l 's, an injective function $\text{int}^{|G_i|} \rightarrow \text{int}$) $*/$;

$l := f_{|G_i|}(t.\text{Rng}, l_{i_1}, \dots, l_{i_k})$;

$U'_j := U'_j \cup \{(G_i, l, t.\bar{T}, t.\bar{A})\}$;

$W' := \bigcup_i \{(g_i, (l_1, \dots, l_m)) \mid G_i = \{c_1, \dots, c_m\} \text{ and } (c_1, l_1), \dots, (c_m, l_m) \in W\}$;

Output (U'_1, \dots, U'_m, W') ;

end

variable, whose domain becomes the product of the domains of the variables from that class. Figure 5 shows a U-relational database and its normalization.

Theorem 4.2. *Given a reduced U-relational database, Algorithm 1 computes a normalized reduced U-relational database that represents the same world-set.*

Computing certain answers. Given a set of possible worlds, we call a tuple certain iff it occurs in each of the worlds. It is known that the tuple certainty problem is coNP-hard for a number of representation systems, ranging from attribute-level ones like WSDs to tuple-level ones like ULDBs [6]. In case of tuple-level normalized U-relations, however, we can efficiently compute the certain tuples using relational algebra.

Lemma 4.3. A tuple \bar{t} is certain in a tuple-level normalized U-relation U iff there exists a variable x such that $(x \mapsto l, \bar{s}, \bar{t}) \in U$ for each domain value l of x and some tuple id \bar{s} .

The condition of the lemma can be encoded as the following domain calculus expression:

$$\text{cert}(U) := \{\bar{t} \mid \exists x \forall l (x, l) \in W \Rightarrow \exists \bar{s} (x, l, \bar{s}, \bar{t}) \in U\}$$

The equivalent relational algebra query on a tuple-level normalized U-relational database $(U[\text{Var}, \text{Rng}, \bar{T}_R, \bar{A}], W)$ is

$$\pi_{\bar{A}}(\pi_{\text{Var}}(W) \times \pi_{\bar{A}}(U) - \pi_{\text{Var}, \bar{A}}(W \times \pi_{\bar{A}}(U) - \pi_{\text{Var}, \text{Rng}, \bar{A}}(U))).$$

| U | D_1 | D_2 | T | A |
|-----|-----------------|-----------------|-------|-------|
| | $c_1 \mapsto 1$ | $c_1 \mapsto 1$ | t_1 | a_1 |
| | $c_1 \mapsto 1$ | $c_2 \mapsto 2$ | t_2 | a_2 |
| | $c_1 \mapsto 2$ | $c_1 \mapsto 2$ | t_2 | a_3 |
| | $c_3 \mapsto 1$ | $c_3 \mapsto 1$ | t_3 | a_4 |
| | $c_3 \mapsto 2$ | $c_3 \mapsto 2$ | t_3 | a_5 |

| W | Var | Rng |
|-----|-------|-----|
| | c_1 | 1 |
| | c_1 | 2 |
| | c_2 | 1 |
| | c_2 | 2 |
| | c_3 | 1 |
| | c_3 | 2 |

(a) U-relational database

| U' | D | T | A |
|------|-------------------------|-------|-------|
| | $c_{12} \mapsto (1, 1)$ | t_1 | a_1 |
| | $c_{12} \mapsto (1, 2)$ | t_1 | a_1 |
| | $c_{12} \mapsto (1, 2)$ | t_2 | a_2 |
| | $c_{12} \mapsto (2, 1)$ | t_2 | a_3 |
| | $c_{12} \mapsto (2, 2)$ | t_2 | a_3 |
| | $c_3 \mapsto 1$ | t_3 | a_4 |
| | $c_3 \mapsto 2$ | t_3 | a_5 |

| W' | Var | Rng |
|------|----------|--------|
| | c_{12} | (1, 1) |
| | c_{12} | (1, 2) |
| | c_{12} | (2, 1) |
| | c_{12} | (2, 2) |
| | c_3 | 1 |
| | c_3 | 2 |

(b) Database from (a) normalized

| c_{12} | $t_1.A$ | $t_2.A$ |
|----------|---------|---------|
| (1, 1) | a_1 | \perp |
| (1, 2) | a_1 | a_2 |
| (2, 1) | \perp | a_3 |
| (2, 2) | \perp | a_3 |

 \times

| c_3 | $t_3.A$ |
|-------|---------|
| 1 | a_4 |
| 2 | a_5 |

(c) WSD corresponding to (b)

Figure 5. Normalization example.

5 Succinctness and Efficiency

This section compares U-relational databases with WSDs [4, 6] and ULDBs [8] using two yardsticks: succinctness, i.e., how compactly can they represent world-sets, and efficiency of query evaluation.

WSDs vs. U-Relations. WSDs are essentially normalized U-relational databases where each variable c_i of a U-relation corresponds to a WSD *component* relation C_i and each domain value l_i of c_i corresponds to a tuple of C_i . Figure 5(c) shows a WSD equivalent to a normalized U-relational database. The normalization may lead to an exponential blow-up in the database size and accounts for U-relations with arbitrarily large ws-descriptors being more compact than U-relations with singleton ws-descriptors and thus than WSDs.

Example 5.1. Consider a relation over schema $R[AB]$ where each field value can be 0 or 1, and $t_i.A$ and the tuple fields $t_{(i+1) \bmod n}.B$ depend on each other ($1 \leq i \leq n$). The encodings as WSD and as a set of two U-relations are given in Figure 6. \square

Theorem 5.2. *U-relational databases are exponentially more succinct than WSDs.*

Positive relational queries have polynomial data complexity for U-relations (Section 3) and exponential data complexity for WSDs [6]. This can be explained in close

| c_1 | $t_1.A$ | $t_2.B$ |
|-------|---------|---------|
| w_1 | 1 | 1 |
| w_2 | 0 | 0 |

 $\times \cdots \times$

| c_n | $t_n.A$ | $t_1.B$ |
|-------|---------|---------|
| w_1 | 1 | 1 |
| w_2 | 0 | 0 |

(a) WSD encoding.

| U_1 | D | T | A |
|-------|-------------------|-------|-----|
| | $c_1 \mapsto w_1$ | t_1 | 1 |
| | $c_1 \mapsto w_2$ | t_1 | 0 |
| | \vdots | | |
| | $c_n \mapsto w_1$ | t_n | 1 |
| | $c_n \mapsto w_2$ | t_n | 0 |

| U_2 | D | T | B |
|-------|-------------------|-------|-----|
| | $c_2 \mapsto w_1$ | t_2 | 1 |
| | $c_2 \mapsto w_2$ | t_2 | 0 |
| | \vdots | | |
| | $c_n \mapsto w_1$ | t_1 | 1 |
| | $c_n \mapsto w_2$ | t_1 | 0 |

(b) U-relational encoding.

Figure 6. WSD and U-relational encoding of the world-set of Example 5.1.

| $c_1 \times \cdots \times c_n$ | $t_1.A$ | $t_2.B$ | \dots | $t_n.A$ | $t_1.B$ |
|--------------------------------|---------|---------|----------|---------|---------|
| w_1 | 1 | 1 | \dots | 1 | 1 |
| w_2 | 0 | 0 | \dots | 0 | 0 |
| w_3 | \perp | 1 | \dots | 1 | \perp |
| | | | \vdots | | |
| w_{2^n} | \perp | \perp | \dots | \perp | \perp |

(a) WSD encoding.

| U_3 | D_1 | D_2 | T | A | B |
|-------|-------------------|-------------------|-------|-----|-----|
| | $c_2 \mapsto w_1$ | $c_3 \mapsto w_1$ | t_2 | 1 | 1 |
| | $c_2 \mapsto w_2$ | $c_3 \mapsto w_2$ | t_2 | 0 | 0 |
| | \vdots | | | | |
| | $c_1 \mapsto w_1$ | $c_n \mapsto w_1$ | t_1 | 1 | 1 |
| | $c_1 \mapsto w_2$ | $c_n \mapsto w_2$ | t_1 | 0 | 0 |

(b) U-relational encoding.

Figure 7. WSD and U-relation representing the answer to $\sigma_{A=B}(R)$ with R of Figure 6.

analogy to the difference in succinctness and by the fact that query evaluation creates new dependencies [10]: U-relations can efficiently store the new dependencies by enlarging ws-descriptors, whereas WSDs correspond to U-relations with normalized ws-descriptors, hence the exponential blowup.

Example 5.3. Consider the WSD and U-relations of Example 5.1 and the selection with join condition $\sigma_{A=B}(R)$. The answer is represented by the WSD and U-relation respectively shown in Figure 7. The U-relation U_3 has $2 \cdot n$ tuples, whereas the WSD $c_1 \times \cdots \times c_n$ has 2^n tuples, each representing a possible combination of the values of the existing fields (a tuple t_i does not occur in worlds where $t_i.A$ or $t_i.B$ have values \perp). Note that by normalizing U_3 we would also obtain one variable with 2^n domain values, as for the WSD.

The answer to $\text{poss}(\sigma_{A=B}(R))$ is efficiently computed as $\pi_{A,B}(U_3)$ in the case of U-relations. In the WSD case, it is computed as $\bigcup_i^n (\pi_{t_i.A, t_i.B}(c_1 \times \cdots \times c_n))$. \square

Finally, the query translations employed by the evaluation algorithms in the WSD and U-relational cases are different. Whereas for WSDs all operators are translated to sequences of relational queries and in the case of projection and join even to fixpoint programs [4], the translation remains strictly in relational algebra for U-relations.

ULDBs vs. U-Relations. A ULDB relation is a set of x -tuples, where each x -tuple represents a set of alternatives. One world is defined by choosing precisely one alternative of each x -tuple. A world may contain none of the alternatives of an x -tuple, if this x -tuple is marked as optional (or *maybe*) using the $?$ -symbol. Dependencies between alternatives of different x -tuples are enforced using *lineage*: An alternative i of an x -tuple s occurs in the same worlds with an alternative j of another x -tuple t if the lineage of (s, i) points either to (t, j) , or to another alternative that transitively points to (t, j) . The lineage of an alternative can also point to an *external* symbol (t, j) , if there is no alternative (t, j) in the database [8].

Example 5.4. The U-relations representing relation R in Figure 1 admit the following equivalent ULDB:

| | R (Id, Type, Faction) | |
|-----|---------------------------|-------------------------------------|
| a | 1: (1, Tank, Friend) | |
| b | 1: (2, Transport, Friend) | 2: (3, Transport, Friend) Λ |
| c | 1: (3, Tank, Enemy) | 2: (2, Tank, Enemy) |
| d | 1: (4, Tank, Friend) | 2: (4, Tank, Enemy) |
| | 3: (4, Transport, Friend) | 4: (4, Transport, Enemy) |

$$\Lambda \text{ is } \lambda(b, 1) = \{(c, 1)\}, \lambda(b, 2) = \{(c, 2)\}$$

To construct an ULDB equivalent to the U-relational database of Figure 1, we have to enumerate all possible value combinations for the attributes of R . This enumeration is not necessary for U-relations because of vertical partitioning and the independence of (most) tuple fields. \square

Lemma 5.5. *ULDBs [8] can be translated linearly into U-relational databases.*

Proof. We sketch the proof for a single ULDB relation R ; it can be extended trivially to the case of several relations.

For every x -tuple t in R we create a new variable c_t , and for each alternative j of t we create a new domain value $w_{(t,j)}$ of c_t . For every alternative in R with value a , id (t, j) and lineage $\lambda(t, j) = \bigwedge_i^{n_{(t,j)}} (t_i, j_i)$ we create a tuple in U^R with value a , tuple id t and ws-descriptor $(n = n_{(t,j)})$

$$D_{(t,j)} = [(c_t, w_{(t,j)}), (c_{t_1}, w_{(t_1,j_1)}), \dots, (c_{t_n}, w_{(t_n,j_n)})].$$

In case $n_{(t,j)}$ is smaller than $n_{(s,l)}$ of an alternative l of an x -tuple s , then we pad the above ws-descriptor with $n_{(s,l)} - n_{(t,j)}$ pairs $(c_t, w_{(t,j)})$.

Q_1 : possible (select o.orderkey, o.orderdate, o.shippriority from customer c, orders o, lineitem l where c.mktsegment = 'BUILDING' and c.custkey = o.custkey and o.orderkey = l.orderkey and o.orderdate > '1995-03-15' and l.shipdate < '1995-03-17')

Q_2 : possible (select extendedprice from lineitem where shipdate between '1994-01-01' and '1996-01-01' and discount between '0.05' and '0.08' and quantity < 24)

Q_3 : possible (select n1.name, n2.name from supplier s, lineitem l, orders o, customer c, nation n1, nation n2 where n2.nation = 'IRAQ' and n1.nation = 'GERMANY' and c.nationkey = n2.nationkey and s.supkey = l.supkey and o.orderkey = l.orderkey and c.custkey = o.custkey and s.nationkey = n1.nationkey)

Figure 8. Queries used in the experiments.

The world table W is the set of pairs of variables and domain values created for the x -tuples of R . For each optional x -tuple t in R , we also add to W a tuple (c_t, w) where w is a fresh domain value for c_t . \square

There are U-relations, however, whose ULDB encodings are necessarily exponential in the arity of the logical relation. This is the case of, e.g., or-set relations [13], attribute-level representations that can be linearly encoded as U-relations but exponentially as ULDBs.

Theorem 5.6. *U-relational databases are exponentially more succinct than ULDBs.*

Both ULDBs and U-relations have polynomial data complexity for positive relational queries. Differently from ULDBs, evaluating queries on U-relations is possible using relational algebra only. The main difference between their evaluation algorithms concerns erroneous tuples, i.e., tuples that do not appear in any world. In contrast to U-relations, erroneous tuples may appear in the answers to queries on ULDBs (see [8] for an example). The removal of such tuples is called data minimization, an expensive operation that involves the computation of the transitive closure of lineage [8]. Such tuples occur with ULDBs because the lineage of an alternative in the answer only points to the lineage of alternatives from the input relations, even though these input alternatives may not occur in the same world. This cannot happen with U-relations because each query operation ensures that only valid tuples are in the query answer by (1) using the ψ -condition in the join and merge operations and by (2) carrying all dependencies in the ws-descriptors – and not only to tuples of the input relation.

To sum up, U-relations have the advantages of WSDs (attribute-level representation) and ULDBs (polynomial evaluation of positive relational algebra queries), while forming an exponentially more succinct representation system than both aforementioned approaches.

| scale | correlation | TPC-H dbsize | #worlds | lworlds | dbsize | #worlds | lworlds | dbsize | #worlds | lworlds | dbsize |
|-------|-------------|--------------|----------------|---------|--------|----------------|---------|--------|--------------------|---------|--------|
| 0.01 | 0.10 | 17 | $10^{857.076}$ | 21 | 82 | $10^{7955.30}$ | 57 | 85 | $10^{79354.1}$ | 57 | 114 |
| 0.01 | 0.25 | 17 | $10^{729.529}$ | 33 | 82 | $10^{6728.24}$ | 129 | 85 | $10^{66995.5}$ | 193 | 118 |
| 0.01 | 0.50 | 17 | $10^{523.031}$ | 71 | 82 | $10^{4724.56}$ | 901 | 88 | $10^{46675.6}$ | 662 | 139 |
| 0.05 | 0.10 | 85 | $10^{4287.23}$ | 22 | 389 | $10^{39913.8}$ | 33 | 403 | 10^{396137} | 65 | 547 |
| 0.05 | 0.25 | 85 | $10^{3633.49}$ | 57 | 389 | $10^{33702.3}$ | 148 | 405 | 10^{334450} | 158 | 567 |
| 0.05 | 0.50 | 85 | $10^{2549.14}$ | 178 | 390 | $10^{23515.5}$ | 449 | 416 | 10^{232650} | 1155 | 672 |
| 0.10 | 0.10 | 170 | $10^{8606.77}$ | 27 | 773 | $10^{79889.9}$ | 49 | 802 | 10^{793611} | 53 | 1090 |
| 0.10 | 0.25 | 170 | $10^{7276.46}$ | 74 | 774 | $10^{67477.1}$ | 145 | 806 | 10^{670090} | 172 | 1132 |
| 0.10 | 0.50 | 170 | $10^{5044.65}$ | 181 | 776 | $10^{46901.8}$ | 773 | 826 | 10^{466038} | 924 | 1339 |
| 0.50 | 0.10 | 853 | $10^{43368.0}$ | 49 | 3843 | 10^{400185} | 71 | 3987 | $10^{3.96845e+06}$ | 85 | 5427 |
| 0.50 | 0.25 | 853 | $10^{36630.3}$ | 130 | 3845 | 10^{337905} | 172 | 4008 | $10^{3.35095e+06}$ | 320 | 5632 |
| 0.50 | 0.50 | 853 | $10^{25528.9}$ | 214 | 3856 | 10^{234840} | 1832 | 4012 | $10^{2.33083e+06}$ | 2586 | 6682 |
| 1.00 | 0.10 | 1706 | $10^{87203.0}$ | 57 | 7683 | 10^{800997} | 99 | 7971 | $10^{7.93774e+06}$ | 113 | 11264 |
| 1.00 | 0.25 | 1706 | $10^{73652.5}$ | 170 | 7687 | 10^{676223} | 208 | 8012 | $10^{6.70229e+06}$ | 344 | 11280 |
| 1.00 | 0.50 | 1706 | $10^{51290.9}$ | 993 | 7712 | 10^{470401} | 1675 | 8228 | $10^{4.66222e+06}$ | 3392 | 13312 |
| | | x = 0.0 | x = 0.001 | | | x = 0.01 | | | x = 0.1 | | |

Figure 9. Total number of worlds, max. number of local worlds in a component, and size in MB of the U-relational database for each of our settings.

6 Experiments

Prototype Implementation. We implemented the query translator of Figure 4 and also extended the C implementation of the TPC-H population generator version 2.6 build 1 [16] to generate attribute and tuple-level U-relations and ULDBs. The code is available on the MayBMS project page (<http://www.infosys.uni-sb.de/projects/maybms>).

Setup. The experiments were performed on a 3GHZ/1GB Pentium running Linux 2.6.13 and PostgreSQL 8.2.3.

Generation of uncertain data. Our data generator creates eight tables: part, partsupp, supplier, customer, lineitem, orders, nation, region. The field values are sensitive to the attribute types and are randomly generated or randomly chosen from the dictionary explained in the TPC-H benchmark specification. The following parameters were used to tune the generation: *scale* (s), *uncertainty ratio* (x), *correlation ratio* (z), and *maximum alternatives per field* (m). The (dbgen standard) parameter s is used to control the size of each world; x controls the percentage of (uncertain) fields with several possible values, and m controls how many possible values can be assigned to a field. The parameter z defines a Zipf distribution for the variables with different dependent field counts³ (DFC) and controls the attribute correlations: For n uncertain fields, there are $\lceil C * z^i \rceil$ variables with DFC i , where $C = n(z - 1)/(z^{k+1} - 1)$, i.e., $n = \sum_{i=0}^k (C * z^i)$. The number of domain values of a variable with DFC $k > 1$ is chosen using the formula $p^{k-1} * \prod_{i=1}^k (m_i)$, where m_i is the number of different values for the field i dependent on that variable and p is the probability that a combination of possible values for the k fields occurs. This assumption fits naturally to data cleaning scenarios. Previous work [4] shows

that chasing dependencies on WSDs enforces correlations between field values and removes combinations that violate the dependencies. We considered here that after correlating two variables with arbitrary DFCs, $100(1 - p)$ percent of the combinations violate constraints and thus are dropped.

The uncertain fields are assigned randomly to variables. This can lead to correlations between fields belonging to different tuples or even to different relations. This fits to scenarios where constraints are enforced across tuples or relations. We do not assume any kind of independence of our initial data as done in several other approaches [10, 8].

Our data generator works as follows. While generating tuples for the eight tables, we use the uncertainty ratio to decide at each tuple field if it is uncertain or not. We collect in a field pool the coordinates (i.e., relation, tuple id, attribute) of the uncertain tuple fields and when the original TPC-H generator finishes its job or the field pool is full, we shuffle the uncertain tuple fields, compute the correlation ratio for variables with different DFC, and incrementally assign tuple fields to variables. Then, we compute the domain size of each variable, and the number of different values for each of variable's fields. The field values are then generated using the data distribution and dictionary for that field type, as specified by the original TPC-H generator. Because there can be too many field coordinates to keep in memory at a time, we use in our experiments a window of 10 million fields to be processed in bulk⁴; after a window is processed, the memory is released, and a new window is filled in and processed. The window size influences the number and dependent field count of the variables. For the experiments, we fixed p to 0.25, m to 8, and varied the remaining parameters as follows: s ranges over

³This is the number of tuple fields dependent on that variable.

⁴It corresponds to a maximum of 500 MB of main memory allocated for dbgen on our testing machine.

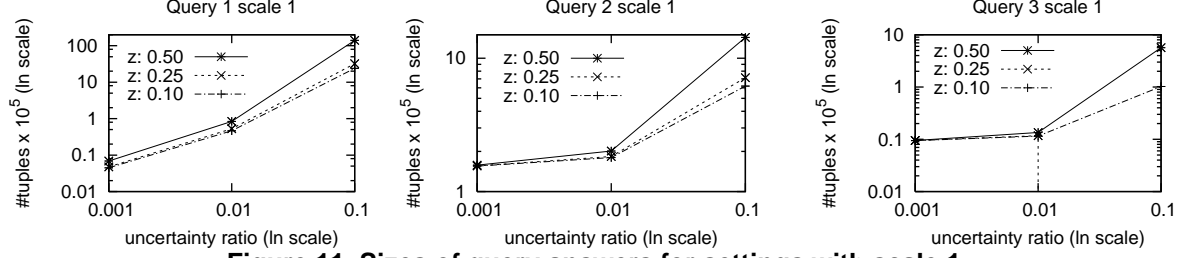


Figure 11. Sizes of query answers for settings with scale 1.

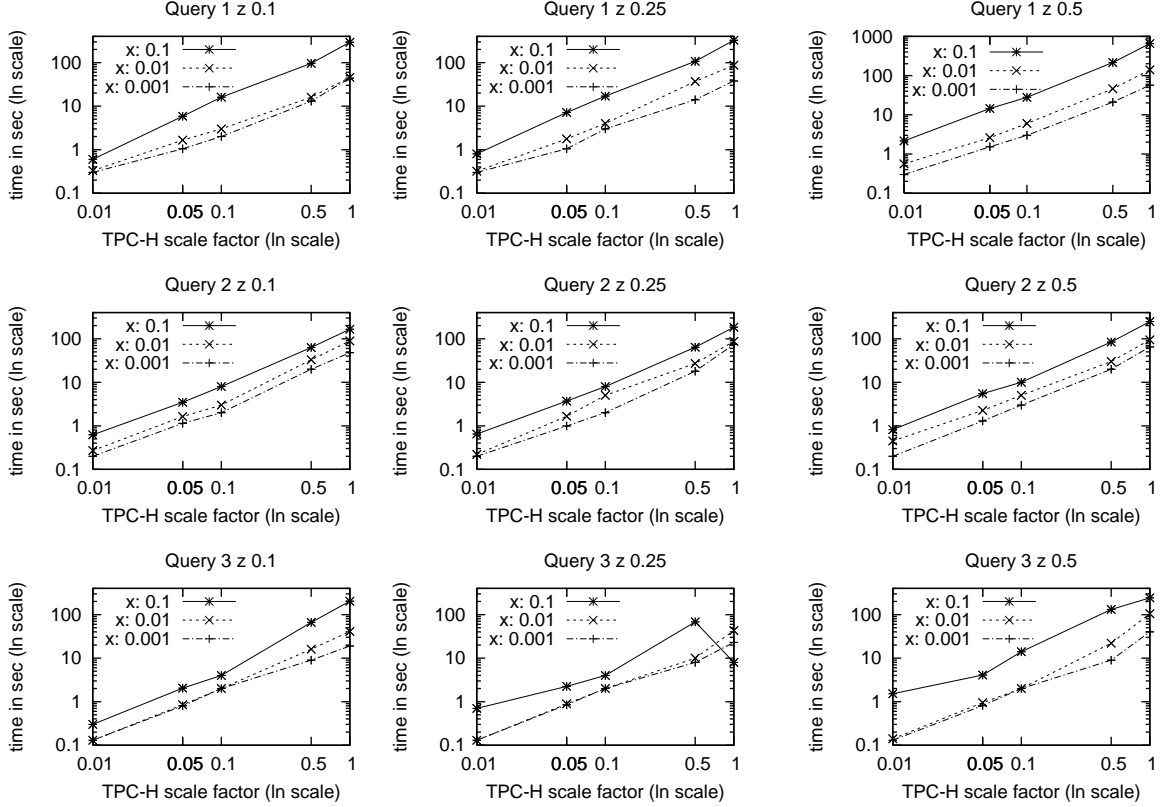


Figure 12. Performance of query evaluation for various scale, uncertainty, and correlation.

The evaluation time varies linearly with all of our parameters. For Q_1 (Q_2 and Q_3 respectively) we witnessed a factor of up to 6 (4 and 10 respectively) in the evaluation time when varying the uncertainty ratio from 0.001 to 0.1. When the correlation ratio is varied from 0.1 to 0.5, the evaluation time increases by a factor of up to 3; this is also explained by the increase in the input and answer sizes, cf. Figures 9 and 11. When the scale parameter is varied from 0.01 to 1, the evaluation time increases by a factor of up to 400; in case of Q_3 and $z = 0.5$, we also noticed some outliers where the increase factor is around 1000. The considerably smaller evaluation time for Q_3 in case of scale 1, uncertainty 0.1, and correlation 0.25 occurs because for that scenario no ‘GERMANY’ entry is generated for the nation table, thus the query answer is empty.

Effect of attribute-level representation. We also performed query evaluation on tuple-level U-relations, which represent the same world-set as the attribute-level U-relations of Figure 9, and on Trio’s ULDBs [8] obtained by a (rather direct) mapping from the tuple-level U-relations. To date, Trio has no native support for the poss operator or the removal of erroneous tuples in the query answer, though this effect can be obtained as part of the confidence computation⁵. For that reason, we decided to compare the evaluation times of queries without the poss operator and without the (expensive) removal of erroneous tuples or confidence computation (which is an exponential-time problem). Since our data exhibits a high degree of (randomly generated) dependency, its ULDB representation has lineage and thus join

⁵Personal communication with the TRIO team as of June 2007.

queries can introduce erroneous tuples in the answer. The Trio prototype was set to use the (faster) SPI interface of PostgreSQL (and not its default python implementation).

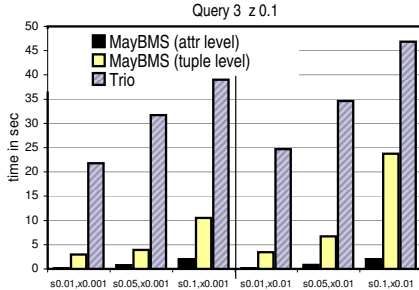


Figure 14. Querying attribute- and tuple-level U-relations in MayBMS and ULDBs in Trio.

Figure 14 compares the evaluation time on attribute- and tuple-level U-relations in MayBMS, and ULDBs for small scenarios of 1% uncertainty, our lowest correlation factor 0.1, and scale up to 0.1. On attribute-level U-relations, the queries perform several times better than on tuple-level U-relations and by an order of magnitude better than ULDBs. This is because attribute-level data allows for late materialization: selections and joins can be performed locally and tuple reconstruction is done only for successful tuples. We witnessed that an increase in any of our parameters would create prohibitively large (exponential in the arity) tuple-level representations. For example, for scale 0.01 and uncertainty 10%, relation lineitem contains more than 15M tuples compared to 80K in each of its vertical partitions.

7 Conclusion and Future Work

This paper introduces U-relational databases, a simple representation system for uncertain data that combines the advantages of existing systems, like ULDBs and WSDs, without sharing their drawbacks. U-relations are exponentially more succinct than both WSDs and ULDBs. Positive relational algebra queries are evaluated purely relationally on U-relations, a property not shared by any other previous succinct representation system. Also, U-relations are a simple formalism which poses a small burden on implementors.

We next briefly report on two current research directions. **Probabilistic U-relations.** U-relational databases can be elegantly extended to model probabilistic information by just adding a probability column P to the world table W . For each variable x , the sum of the values $\pi_P(\sigma_{\text{Var}=x}(W))$ must equal one. We can then assign probability to any subset of the world-set, described by a ws-descriptor \bar{d} , as the product of probabilities of each variable assignment in \bar{d} .

The techniques for evaluating the operations of positive relational algebra presented in this paper are applicable in the probabilistic case without changes. Computing the confidences of the answer tuples is an inherently hard problem [10]. Our current research investigates practical approximation techniques for confidence computation.

Support for new language constructs. Following our recent investigation on uncertainty-aware language constructs beyond relational algebra [5], we identified common physical operators needed to implement many primitives for the creation and grouping of worlds. It appears that normalizing sets of ws-descriptors in the sense of Section 4 plays an important role in evaluating these operations and in confidence computation. We are currently working on secondary-storage algorithms for normalization.

References

- [1] S. Abiteboul, P. Kanellakis, and G. Grahne. “On the representation and querying of sets of possible worlds”. *Theor. Comput. Sci.*, **78**(1), 1991.
- [2] P. Andritsos, A. Fuxman, and R. J. Miller. “Clean Answers over Dirty Databases: A Probabilistic Approach”. In *Proc. ICDE*, 2006.
- [3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. “Fast and Simple Relational Processing of Uncertain Data”. Technical Report INFOSYS-TR-2007-2, Saarland University, 2007.
- [4] L. Antova, C. Koch, and D. Olteanu. “ 10^{106} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information”. In *Proc. ICDE*, 2007.
- [5] L. Antova, C. Koch, and D. Olteanu. “From Complete to Incomplete Information and Back”. In *Proc. SIGMOD*, 2007.
- [6] L. Antova, C. Koch, and D. Olteanu. “World-set Decompositions: Expressiveness and Efficient Algorithms”. In *Proc. ICDT*, 2007.
- [7] D. S. Batory. “On Searching Transposed Files”. *ACM Trans. Database Syst.*, **4**(4):531–544, 1979.
- [8] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. “ULDBs: Databases with Uncertainty and Lineage”. In *Proc. VLDB*, 2006.
- [9] R. Cheng, S. Singh, and S. Prabhakar. “U-DBMS: a database system for managing constantly-evolving data”. In *Proc. VLDB*, 2005.
- [10] N. Dalvi and D. Suciu. “Efficient query evaluation on probabilistic databases”. In *Proc. VLDB*, 2004.
- [11] G. Grahne. “Dependency Satisfaction in Databases with Incomplete Information”. In *Proc. VLDB*, 1984.
- [12] T. Imielinski and W. Lipski. “Incomplete information in relational databases”. *Journal of ACM*, **31**(4), 1984.
- [13] T. Imielinski, S. Naqvi, and K. Vadaparty. “Incomplete objects — a data model for design and planning applications”. In *Proc. SIGMOD*, 1991.
- [14] P. Sen and A. Deshpande. “Representing and Querying Correlated Tuples in Probabilistic Databases”. In *Proc. ICDE*, 2007.
- [15] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O’Neil, P. E. O’Neil, A. Rasin, N. Tran, and S. B. Zdonik. “C-Store: A Column-oriented DBMS”. In *Proc. VLDB*, 2005.
- [16] Transaction Processing Performance Council. *TPC Benchmark H (Decision Support)*, revision 2.6.0 edition, 2006. <http://www.tpc.org/tpch/spec/tpch2.6.0.pdf>.